UTILITIES

TIME!: A Threefold Digital Timer for Windows

BY DOUGLAS BOLING

Through its built-in controls, the Windows interface delivers bonuses to both users and programmers. Users benefit because every control within an application works in a friendly and familiar way—radio buttons, check boxes, and list boxes, to name a few. The ease of use and time savings come about with relatively little development investment, too. Programmers no longer need to wrestle with gruesome implementation details to create a slick, serviceable interface, thanks to these Windows controls supplied by Microsoft. Disparate applications also gain a common look because developers find these standard tools so much easier to employ than roll-your-own code. Again, end users reap the rewards.

Windows promotes standardization, but allows for some extension by providing programmers with the means to create custom controls specifically designed for an application. TIME!, this issue's utility, demonstrates the use of these controls, while simultaneously delivering a useful program.

You can download TIME!.EXE and the source code from Library 2 (PCMag Utils) of the Utilities/Tips Forum on PC MagNet archived as TIME!.ZIP. If you just want the files you need to run TIME!, download the self-extracting archive TIME.EXE. For instructions, refer to the sidebar "Utilities by Modem." If you don't have a modem, you can obtain the files directly from *PC Magazine* by sending a postcard with your name, address, and preferred disk size to the attention of Katherine West, *PC Magazine,* One Park Ave., New York, NY 10016. Or you can fax your request to 212-503-5799 (no phone calls please). Compiling TIME! requires Borland C++ 3.0, or Microsoft C 6.0 and the Windows 3.*x* SDK (Software Development Kit).

USING TIME! TIME! works the way the cooking timer in your kitchen does. The program actually contains three independent timers, each with the ability to be set, started, and stopped. All share a single display, however, so only one timer's setting can appear during any given period; the three buttons immediately below the display let you make your selection. Pressing the Timer 1 button, for example, instructs the program to show time remaining on its first timer. A black bar above the button indicates the timer you've chosen.

TIME!, this issue's utility, is a triple digital timer that shows programmers how to implement custom controls in Windows.

The three timers can run simultaneously. The digital display shows the time remaining until the currently selected timer reaches zero. Figure 1 shows the TIME! window.

The Minute and Second buttons allow you to set the time. To add a minute to the currently displayed timer, you simply press the Minute button. The Second button works the same way. The buttons function only if the timer is stopped.

The Start/Stop button performs its functions on the displayed timer only. If the timer is running, the button is labeled Stop; otherwise it says Start. A timer can be started and stopped repeatedly. When restarted, the count continues from the currently displayed time. For example, if you halt a timer that is showing 0:59, add one minute, and reactivate it, counting continues from the new time—1:59.

A shortcut allows you to quickly set the displayed time with a minimum of effort. Clicking the left mouse button on a digit increases it by one. Clicking the right button decreases it a unit. Again, the timer must be stopped.

After you've set and started one or more timers, you can minimize the TIME! window. When one of the timers expires, TIME! pops to the foreground and beeps the speaker.

The look of the TIME! digits can be set to resemble either a liquid crystal display (LCD) or a light-emitting diode (LED) display. In LCD mode, TIME! displays the digits as black numbers on a gray background (like LCD watches). LED mode produces red numbers on a black background. While LED watches went out of style with Disco, red on black still looks pretty good.

Changing the look is simple. Choose either LED or LCD from the Display menu. Finally, to terminate the program, choose Exit.

To install TIME!, copy the files

Utilities by Modem

The *PC Magazine* utilities are available by modem from PC MagNet, a ZiffNet service hosted by CompuServe.

To find the phone number nearest you, set your communications software to 300, 1,200, 2,400, or 9,600 bits per second, 7 data bits, even parity, 1 stop bit, and full duplex, then dial 800-346-3247 with your modem. When the modem connects, press Enter. At the HOST NAME prompt, enter PHONES. Follow the menus and note the number closest to you. Or you can call 800-635-6225 (voice) and follow the instructions and note the number.

To obtain the current issue's utility free of charge, simply dial the local number; at the HOST NAME prompt, type CIS; and at the USER ID prompt, enter 60116,1. Then, at the PASSWORD prompt, enter PC-MAGUTIL.

To join ZiffNet, at the USER ID prompt, type 177000,5000. Then, at the PASSWORD prompt, enter PC* MAGNET. Finally, at the ENTER AGREEMENT NUMBER prompt, type PCMAG92.

Register your name and enter your American Express, MasterCard, or Visa number. (If you'd like to have your company billed instead, call CompuServe at 800-848-8990.) Your personal user ID and a temporary password will be displayed. A new password will arrive in the mail within 10 days to confirm your subscription.

ZiffNet membership costs $2.50 per month. This includes access to *PC Magazine* Editors' Choice Awards, Product Reviews index, Weekly News from *PC Week*, Buyers' Market, ZiffNet Highlights, and the Support Forum (which also includes the current utility). CompuServe members can join by entering GO PCMAG at any CompuServe ! prompt.

Outside of these areas, PC MagNet costs $6.30 per hour for 300-bps service, $12.80 for 1,200 or 2,400 bps, and $22.80 for 9,600 bps. Billing is based on 1-minute increments.

These programs can be copied but are copyrighted. You may make copies for others as long as no charge is involved, but making copies for any commercial purpose is prohibited. □

TIME!.EXE and DIGIT.DLL into a directory located on your path. As with all Windows programs, TIME! can be run from the Program Manager using the File, Run menu selection. To make running the program convenient, you can install the TIME! icon in the Program Manager using the File New menu selection.

WHAT MAKES IT TICK? TIME! is a simple Windows application constructed (as are many Windows programs) with a non-modal dialog box as its main window. This eliminates the need to use the Windows API to create the controls TIME! needs. The size and position of the controls are specified by the dialog-box template in the program's resource script shown in Figure 2. The resource compiler is used to compile the script file. The resulting file, TIME!.RES, must then be bound into the TIME!.EXE file.

While Windows supplies the standard button controls used by TIME!, it provides nothing that can show a number in the form of an LCD or LED digit. These are displayed using a custom control called DigitClass, which we'll examine in detail later. Right now, let's look at how to place the control using a dialog box template.

The CONTROL statements in the dialog box template specify the DigitClass controls. The ones digit of the seconds display, for example, is described by the line

```
CONTROL " ", IDD_SECDIG,
 "DigitClass", DIGS_LED |
 WS_CHILD, 95, 5, 30, 40
```

where CONTROL tells Windows to create a control of type DigitClass within the dialog box at the x,y coordinates 95, 5 and with a size of 30, 40. As with all controls, the quantities are specified in dialog units based on the size of the Windows system font. The label IDD_SECDIG refers to the #define statement in TIME.H that equates the string IDD_SECDIG to 100, the ID tag for this particular control.

DIGS_LED and WS_CHILD are the defined styles for the control. DIGS_LED is specific to DigitClass and indicates that the digit should initially display in red on black. WS_CHILD is a standard Windows style for all window controls and denotes that the control is a child window.

The window procedure for the parent window, WinMain, monitors and commands all of TIME!'s controls. This includes the standard Windows controls (such as buttons and static window classes) as well as the custom DigitClass controls. The procedure is simple, reflecting the simple nature of TIME!: While it does supervise the various controls, WinMain mostly manages the timers.

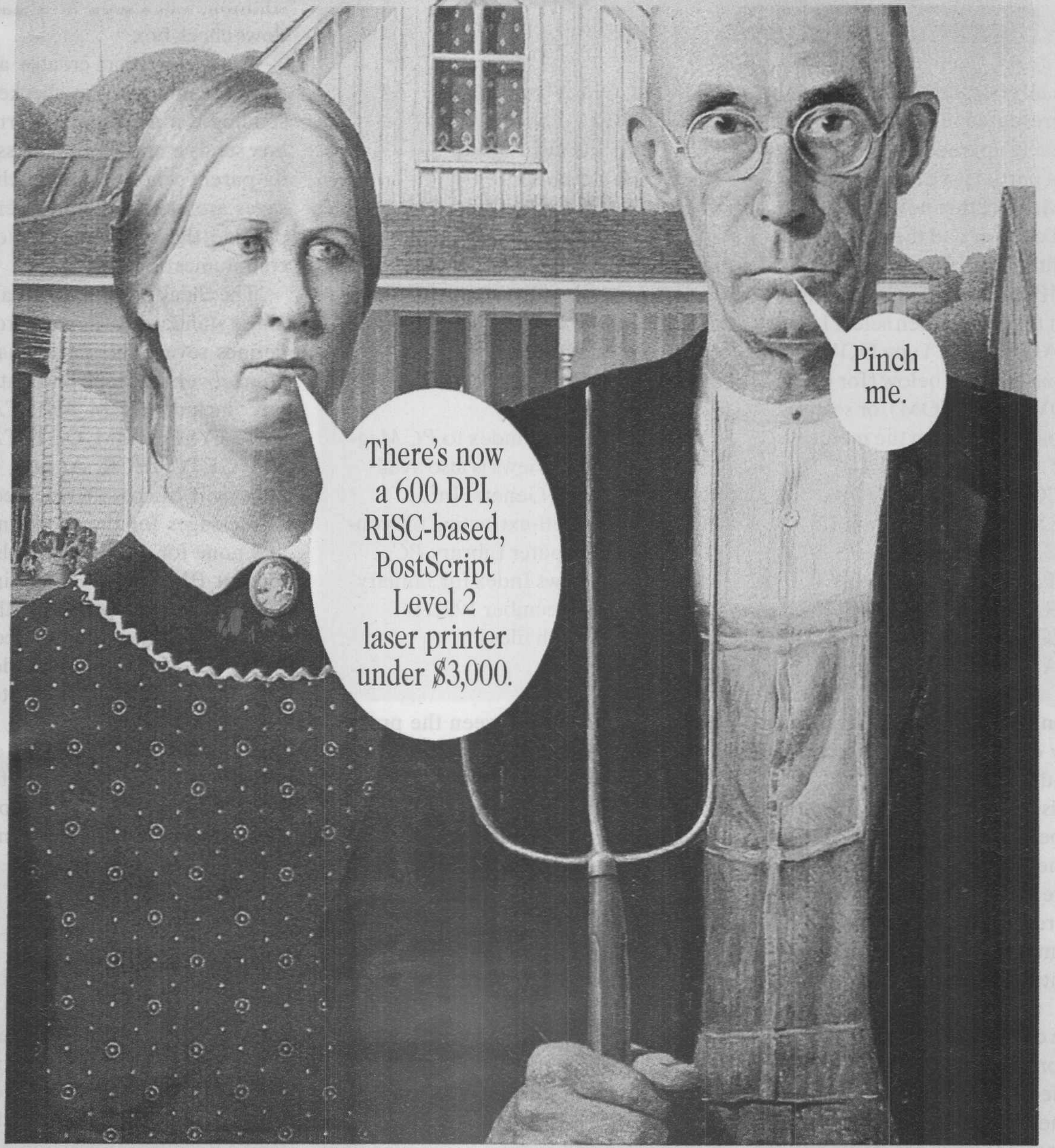
The program's timer function uses just one Windows system timer for all three of its own. The number of Windows timers is limited, so sharing reduces the load on the system. To further lighten demand, the utility only requests a system timer when one of its own runs.

TIME! calls Windows' SetTimer() function to configure a system timer sc that it sends a WM_TIMER message once a second. Upon receipt of each such message, the program calls its ComputeTime routine to determine the period elapsed since the displayed timer was started. The DigitClass controls then show the seconds remaining. A call to KillTimer releases the Windows timer when the utility's last timer expires.

The rest of TIME! is straightforward. The construction of the DigitClass custom control and its use are worth looking at, so I'll take the time to elaborate on those subjects now.

WINDOWS CUSTOM CONTROLS Before implementing a custom control, you should ask yourself if you really need it. Windows' standard interface is immensely flexible. It includes numerous controls for both collecting user input and presenting results. Why should a programmer change this interface with a nonstandard control?

The digits in TIME! could have been realized either as a custom control or simply an embedded window procedure. A control is nothing more than a predefinea window class packaged in a DLL (dynamic link library). The classic Windows


There's now a 600 DPI, RISC-based, PostScript Level 2 laser printer under $3,000.
Pinch me.

The new LZR 965 from Dataproducts gives you everything you want from a laser printer at a price that may startle you. This true 600 x 600 dpi, Adobe® PostScript® Level 2 laser printer gives you greatly enhanced gray scale capabilities for optimum detail in graphics and photographic images. Even 4-point type is made clean and easy to read. And since the RISC-based processor can provide fast, flawless delivery of higher resolution images your output can look like a masterpiece every time. In no time.

	List Price	PPM	RISC Processor	Postscript Level 2	Resolution 600x600	Warranty
LZR 965	$2,995	9	Yes	Yes	Yes	2 years
IBM 4029-10A	$3,995	10	No	No	Yes	1 year
Apple LaserWriter IIg	$4,999	8	No	Yes	No	1 year
QMS PS815MR	$5,495	8	No	No	Yes	1 year

Combine Mac and PC system compatibility and superior paper handling with a very small footprint and you'll begin to experience the whole picture.

You don't need a trained eye to see that the extraordinary image quality and technologically advanced features will make this printer a classic for years to come. Add the fact that it's easy to use and inexpensive to own and you'll know why the LZR 965 has everybody talking. For finer details, contact the Dataproducts dealer nearest you or call 800-334-3174.

Dataproducts.

The high performance printer people.™

CIRCLE 298 ON READER SERVICE CARD

▪ PC Magazine Utilities Updates

As with all good software, the programs presented in *PC Magazine* are periodically upgraded and improved. Here's a partial list of the programs on PC MagNet that have been updated. To download these files from the Utilities/Tips Forum, type GO ZNT:TIPS, type LIB or select Libraries from the menu, then select Library 2 PCMAG UTILS. Type DOW and the filename listed below (for example, DOW ANSI.COM), or select Download a File from the menu.

ADDIT.COM, Version 1.1
ANSI.COM, Version 1.3
BAT2EXEC.COM, Version 1.5
BCOPY, Version 1.2
CARDFILE.COM, Version 1.1
CHKFRAG.EXE, Version 1.7
CONFIG.CTL, Version 3.0
DIRMATCH.COM, Version 3.1
EMMA.COM, Version 2.2
EMS40.SYS, Version 1.1
PCACCESS.EXE, Version 1.1
RN.COM, Version 3.0
SLICE.COM, Version 1.3
SNIPPER.COM, Version 1.2
ZCOPY.COM, Version 1.4

For a complete list of the programs that are available on PC MagNet, download PCMCAT.ZIP from Library 0 (New Uploads) of the Utilities/Tips Forum.

A downloadable index to *PC Magazine*'s product reviews is also available in Library 1 (General Info). PCM.EXE is a self-extracting file containing the Computer Library *PC Magazine* Reviews Index for January 1988 through December 1991. It requires the search files in PCSRCH.EXE.

push button, for example, could be implemented by each Windows program as a simple window procedure.

Controls hold two main advantages over embedded window procedures, though. The most obvious is that all programs have access to the control rather than each requiring its own window procedure. A more subtle but equally important advantage is that isolating the window class in a control allows you to change the control's logic without recompiling the program.

Your decision about using a custom control, then, should be based on portability across different programs and the likelihood of a change in the logic. If you plan to employ the control in a number of applications or, to facilitate future changes, you want to isolate its logic from the rest of the program, a custom control is the answer. If, however, you only need a special window class for one program, then it may not be worth your trouble to implement the window as a custom control. In the case of the DigitClass control, the decision could have gone either way. I do a large amount of Windows programming, though, so I felt that it might come in handy later.

Once you've decided to make a custom control, how to design the communication mechanism between the program and the control will be your first problem. The control will be isolated from the rest of the code, so it can't just share global variables! Examining the predefined controls bundled with Windows reveals the solution. Let's look at a standard Windows check box.

A window that creates a check box control can mark it as checked simply by sending it a message. In turn, the check box sends a notification message back to its parent when it's been clicked. Messages are easy and convenient, and they provide the simplest form of Windows communication.

The check box is actually a special case of the standard Button control. Windows defines several messages that let a program govern the look and state of the button, including BM_SETSTATE, BM_SETSTYLE, BM_GETSTATE, and BM_GETCHECK. At first, it might appear as if Microsoft reserved a number of messages for the predefined controls and none for custom controls. However, look at BM_GETCHECK in the WINDOWS.H file supplied with the Windows SDK or Borland's C++ 3.0 compiler and you'll see how Microsoft designed the control messages so that custom controls can use them too.

BM_GETCHECK is defined as WM_USER+0. Starting at the messag number WM_USER, Windows reserves a large number of Windows messages for

▪ Using the DigitClass Control

Custom controls can be conveniently reused when you program new Windows apps, and this makes them powerful tools. Both the Dialog Box editor in the Microsoft Windows SDK and the Resource Workshop bundled with Borland's C++ 3.1 compiler allow custom controls to be used as if they were part of the standard Windows controls. Let's look at how to use the DigitClass control with the SDK.

Start the SDK's dialog editor and select File Open Custom Control from the menu. A dialog box will appear and you can enter the path to DIGIT.DLL and the DigitClass control will automatically be loaded.

To put the control in a dialog box, select the custom icon in the control toolbox. The pointer will change to a box as you move it back over the outline of the new dialog box. Click the mouse inside the new dialog box to place the control. If you have more than one custom control installed in it, the editor will ask which you want when you press the custom icon.

Using custom controls with Borland's Resource Workshop is just as simple. Select Options Install Custom Control Library from the menu to install the control and choose Controls Custom to place it. Then add the DIGIT.H file to the list of #include files used by the project by selecting the Add to Project menu. Since the Resource Workshop comes with a number of custom controls, you will always be asked which you wish to use.

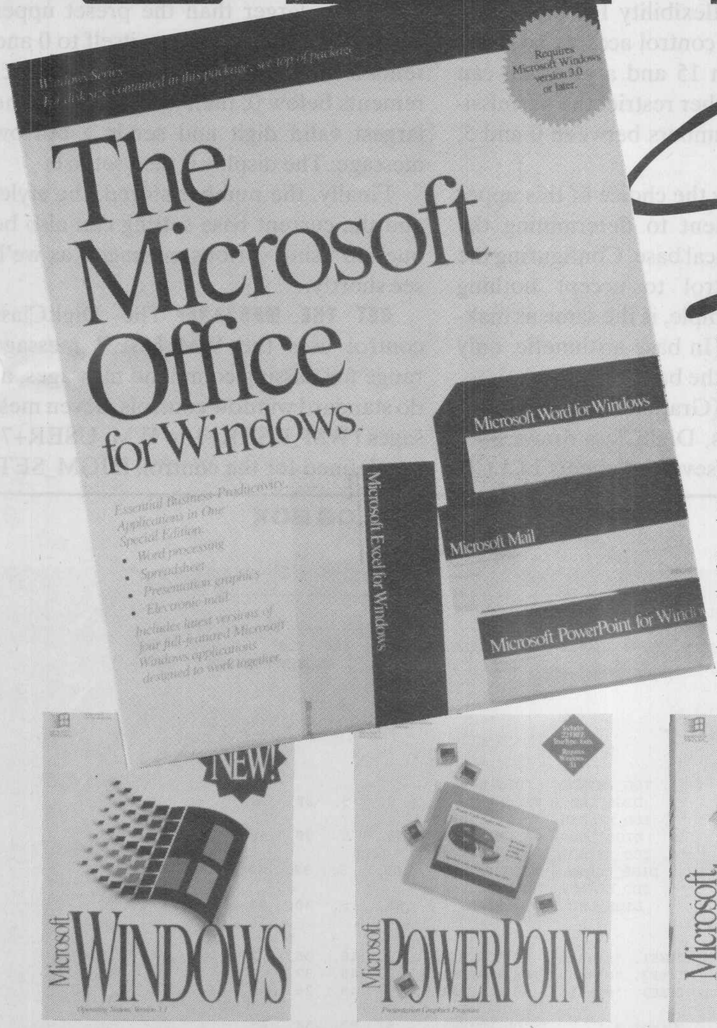
Once installed, it's as simple to use a custom control as any of the standard controls. With custom controls, though, your program takes on a look of its own.—*Douglas Boling*

MICROSOFT

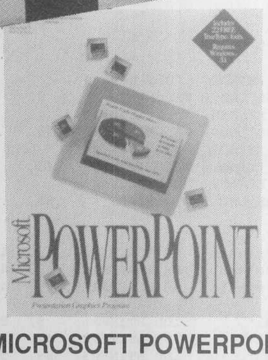
OFFICE

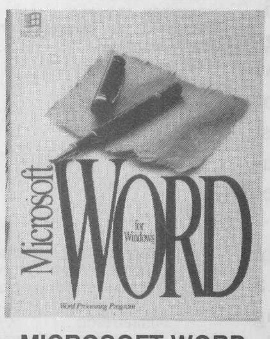
DOES IT RIGHT!

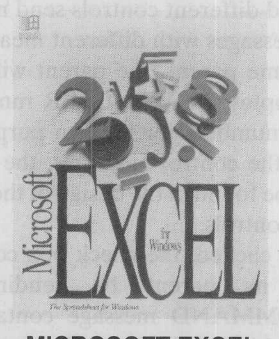
$459


Requires Microsoft Windows version 3.0 or later.
The Microsoft Office for Windows
Essential Business-Productivity Applications in One Special Edition
Microsoft Word for Windows
Microsoft Excel for Windows
Microsoft Mail
Microsoft PowerPoint for Windows

Microsoft does it right by bringing together full-featured Windows versions of Word, Excel, PowerPoint and Mail in one convenient package. And Software Spectrum does it right too by bringing Microsoft Office for Windows to you at a great price—only $459. What's more, we've got low prices on other products from Microsoft. Plus the best service in the industry. **And that does it. Right!**







MICROSOFT WINDOWS	**MICROSOFT POWERPOINT**	**MICROSOFT WORD**	**MICROSOFT EXCEL**
Version 3.1	Version 3.0 for Windows	Version 2.0 for Windows	Version 4.0 for Windows
Only **$95**	Only **$299**	Only **$299**	Only **$299**

To Order Call

1-800-824-3323

7:00 am to 7:00 pm (Central)

Prices Subject To Change.

SOFTWARE SPECTRUM

CIRCLE 261 ON READER SERVICE CARD

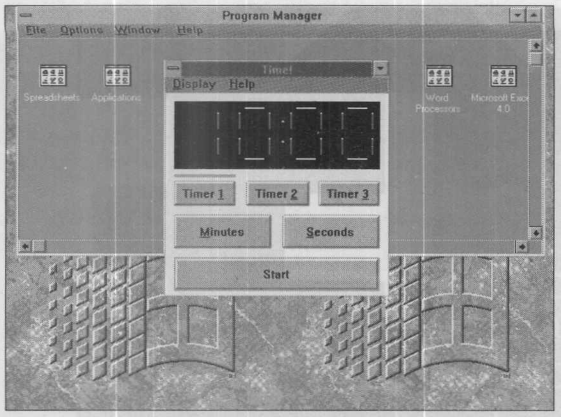


***Figure 1:* This figure shows the TIME! window with the four DigitClass custom controls.**

window procedures to use as they see fit. One twist seems to foul the works, though: The standard controls also employ this range of WM_USER message numbers for command messages. EM_GETSEL, used by the edit control, for instance, is also defined as WM_USER+0.

Since a number of controls use the same message number for different commands, what prevents chaos? In the case of the example above, since you send the BM_GETCHECK message to a check box and EM_GETSEL to an edit control, each control remains unaffected by the other's use of the same message number.

Just as messages command a control, they also are issued when a control must notify its parent window of an event. In this case, though, the neat trick of using numbers that are offsets from the WM_USER message won't work. Not only could different controls send notification messages with different meanings to the same parent, the parent window might employ the WM_USER range of message numbers for its own purposes. As with the control messages, the solution can be found in the design of the predefined controls.

When clicked, the check box control notifies its parent by sending a WM_COMMAND message containing lParam, which holds the notification code in its upper word and the handle of the check box window in its lower word. The ID number of the control, set when the control was created, is found in wParam. Using unique IDs for each of its child window controls, the parent determines precisely which one sent the message and with that, figures out the meaning of the notification code in lParam. The WM_COMMAND notification system is used by all of Windows' standard controls.

THE DIGITCLASS CONTROL DigitClass—a rather simple control—displays a number between 0 and 15, representing those above 9 in hexadecimal fashion, where the letter A corresponds to 10, B to 11, and so on. (Actually, TIME! never shows hex numbers, but including this ability gives DigitClass more flexibility for future applications.) The control accepts no number greater than 15 and a program can instruct it to further restrict the permissible range (to numbers between 0 and 5, for example).

You can view the choice of this upper limit as equivalent to determining the control's numerical base. Configuring the DigitClass control to accept nothing above 9, for example, is the same as making its base 10. (In base arithmetic, only numbers below the base are valid.)

Using GDI (Graphics Device Interface) commands, DigitClass draws lines to simulate the seven segments of LCD and LED displays. Depending on the control's style setting, a decimal poin and a colon can be presented in addition to the number, and the colors can be set to either the LCD or LED mode.

The control also remembers the displayed numeral; once set, the DigitClass control will continue to show it until instructed by another command to change the display.

The number displayed by a DigitClass control can be incremented or decremented simply by sending a message. Normally, when the control receives an increment command, it advances the stored number and displays it. If a number grows larger than the preset upper limit (its base), then it sets itself to 0 and sends a carry message. If the number decrements below 0, then it sets itself to the largest valid digit and sends a borrow message. The display is then set to 0.

Finally, the number stored, the style, and the current base setting can also be queried using various messages, as we'll see shortly.

GET THE MESSAGE? The DigitClass control uses the WM_USER message range for passing command messages, a do standard window controls. Seven messages (WM_USER+1 to WM_USER+7) are defined for the control: DIGM_SET,

MAIN WINDOW DIALOG BOX

Complete Listing

```
//----------------------------------------------------------------------
//  Main window Dialog box
//----------------------------------------------------------------------
WinTime DIALOG LOADONCALL MOVEABLE DISCARDABLE 100, 122, 130, 120
STYLE WS_BORDER ¦ WS_CAPTION ¦ WS_DLGFRAME ¦ WS_SYSMENU ¦ WS_MINIMIZEBOX

CLASS "Time!"
CAPTION "Time!"
FONT 8, "MS Sans Serif"
MENU WinTime
BEGIN
    CONTROL         "", IDD_TMINDIG, "DigitClass",
                     DIGS_LED ¦ WS_CHILD,        5,   5,  30,  40
    CONTROL         "", IDD_MINDIG, "DigitClass",
                     DIGS_LED ¦ WS_CHILD,       35,   5,  30,  40
    CONTROL         "", IDD_TSECDIG, "DigitClass",
            WS_CHILD ¦ DIGS_COLON ¦ DIGS_LED,      65,   5,  30,  40
    CONTROL         "", IDD_SECDIG, "DigitClass",
                     DIGS_LED ¦ WS_CHILD,       95,   5,  30,  40


    CONTROL "",IDD_MTIME1, "static", SS_BLACKRECT,  5,  48,  36,   2
    CONTROL "",IDD_MTIME2, "static", SS_BLACKRECT, 47,  48,  37,   2
    CONTROL "",IDD_MTIME3, "static", SS_BLACKRECT, 89,  48,  36,   2

    PUSHBUTTON      "Timer &1",      IDD_BTIMER1,  5,  52,  36,  15
    PUSHBUTTON      "Timer &2",      IDD_BTIMER2, 47,  52,  37,  15
    PUSHBUTTON      "Timer &3",      IDD_BTIMER3, 89,  52,  36,  15

    PUSHBUTTON      "&Minutes",     IDD_BMINUTES,  5,  72,  57,  20
    PUSHBUTTON      "&Seconds",     IDD_BSECONDS, 68,  72,  57,  20

    PUSHBUTTON      "S&tart",             IDD_GO,  5,  97, 120,  20
END
```

***Figure 2:* This is the listing of the dialog-box template for the main window of Time!. The four statements starting with the word CONTROL are the lines that specify the instances of the DigitClass control.**

"ALPHA FOUR IS INDISPENSABLE! I CAN DEVELOP POWERFUL RELATIONAL DATABASE APPLICATIONS EVEN THOUGH I'M A NON-PROGRAMMER."

— Bruce Berenson, Vice-President; Belgacom, USA

Bruce Berenson, Vice-President of Belgacom, USA, uses a custom-designed Alpha Four database everyday. He tried dBASE® (too complicated) and Q&A® (not powerful enough). But with Alpha Four, Bruce easily designed a professional application to suit his needs. It's the one fully-relational database that empowered him to create a completely customized application — without programming! Now Bruce, and his co-workers, are among the over 400,000 satisfied Alpha Four customers worldwide.

Alpha Four is the only relational database specifically designed for non-programmers who need serious database solutions.

Alpha Four's menu-driven interface is so intuitive, you almost forget that you're working with a powerful, relational database. With lightning fast indexes, professional quality applications, dialog boxes, conditional logic mail merge, mail labels and powerful report writing, Alpha Four fulfills all your database needs. And, Alpha Four uses the standard dBASE file format so you never have to worry about compatibility.

PC Week gave Alpha Four a #1 Rating and *PC Magazine (Feb. 25, 1992)* states, "Alpha Four Version 2 delivers on the promise to be even more powerful without giving up any of the features that earned it our Editor's Choice."

For information, a FREE demo disk, or to order, call 1-800-852-5750, ext. 117.

Alpha Software Corporation
168 Middlesex Turnpike
Burlington, MA 01803 USA
617-229-2924; fax: 617-272-4876
Available through dealers worldwide.

QFS

According to IDC, Alpha Four is the fastest growing, best-selling relational database for non-programmers.

PC/COMPUTING **Best Value**

INFO WORLD EXCELLENT VALUE

DBMS 1991 READERS' CHOICE

PC WORLD BEST BUY

THE PC/COMPUTING 200 BEST PC PRODUCTS 1992

InfoWorld, May 11, 1989; DBMS, December, 1991; PC World, March, 1990; PC Computing, June, 1992, August, 1992
® All trademarks registered to the appropriate companies.

CIRCLE 196 ON READER SERVICE CARD

DIGM_GET, DIGM_INC, DIGM_DEC, DIGM_SETBASE, DIGM_GETBASE, and DIGM_BLANK. Since programs that use the DigitClass control need to know which message number goes with which command, the numbers are assigned in the separate file DIGIT.H, which is listed in Figure 3.

DIGM_SET sets the number that the control will display. The number is passed in the message's wParam (lParam is not used). The number can be between 0 and 15 as long as it does not exceed the base of the control. If a program attempts to set a number above the base, the DigitClass control sets the number to one less than the Base.

The DIGM_SETBASE message sets the numeric base, with the base passed in wParam. Like DIGM_SET, the DIGM_SETBASE message does not use lParam. Since 15 is the highest number allowed, 16 is the maximum base value. If a program attempts to set it higher, the control forces it to 16. The minimum possible base is 2.

DIGM_INC advances the displayed number and DIGM_DEC decrements it. These messages use neither wParam nor lParam.

DIGM_BLANK instructs TIME! to blank the display. DIGM_GET and DIGM_GETBASE let the parent window query the control about the number displayed and the base, respectively. These three messages use neither wParam nor lParam.

No messages exist that govern the display of the decimal and colon but they, as well as colors, can be managed by setting the style bits for the control. The Windows API function calls, GetWindowLong and SetWindowLong, accomplish the task.

The DigitClass control uses the same WM_COMMAND notification scheme as the standard controls, employing one of four notification codes: DIGN_CARRY, DIGN_BORROW, DIGN_LCLICK, and DIGN_RCLICK. As with the standard Windows controls, the codes are sent in the high word of lParam (in a WM_COMMAND message). The low word of lParam holds the control window handle and wParam carries the control ID.

The notification codes themselves are fairly self explanatory. DIGN_CARRY is sent when the DigitClass control is incremented past its base. If, for example, the base has been set to 6 and a DIGM_INC message is forwarded to the control (currently displaying a 5), the control will display a 0 and post a DIGN_CARRY notification to its parent. DIGN_BORROW is posted if the control is decremented below 0. In this case, DigitClass will display one less than the base and then send the DIGN_BORROW notification.

Clicking on the DigitClass control sends DIGN_LCLICK if caused by the left mouse button or DIGN_RCLICK if caused by the right. More specifically, the notification is sent when a DigitClass window receives a WM_LBUTTONUP or WM_RBUTTONUP message.

CONSTRUCTING A CONTROL DigitClass is implemented in a DLL (DIGIT.DLL), as are all Windows controls. Bundling the code in a DLL, however, isn't the only overhead incurred when writing a control; a number of required routines must be added to integrate it into the various dialog-box editors that developers use. In the case of the DigitClass control, this involves as much work as writing its window procedure!

As with any DLL, DIGIT.DLL contains a LibMain procedure called when the DLL first loads into memory. The procedure is comparable to the WinMain function called when a Windows program executes. In DigitClass, LibMain registers the DigitClass window class. This allows programs to create DigitClass controls by simply loading DIGIT.DLL and calling one of the window-creation functions (such as CreateWindow).

The next series of dynamic link library procedures are specific to custom-control DLLs and follow the naming scheme recommended by Microsoft. The names of the routines—DigitWndFn, DigitInfo, DigitStyle, DigitDlgFn, and DigitFlags—are customized for the DigitClass control. Following this guideline, procedure names for a custom control called Wigit, for example, would be WigitWndFn, WigitInfo, WigitStyle, and so forth.

The routines are referenced by their ordinal numbers. The ordinal number for a procedure is its position in the list of callable routines in the DLL header.

DigitWndFn is the window procedure

DIGIT.H
Complete Listing

```
//----------------------------------------------------------------------
//  DIGIT.H -- Include file for DIGIT.DLL and the DigitClass Control
//
//  (c) Douglas Boling, 1992
//----------------------------------------------------------------------
//
// Digit Control Styles
//
#define DIGS_LED          0x0001L
#define DIGS_BLANK        0x0002L
#define DIGS_COLON        0x0004L
#define DIGS_DECIMAL      0x0008L
//
// Digit Contol Style labels (These strings must match above defines.)
//
#define  DIGS_LEDSTR        "DIGS_LED"
#define  DIGS_BLANKSTR      "DIGS_BLANK"
#define  DIGS_COLONSTR      "DIGS_COLON"
#define  DIGS_DECIMALSTR    "DIGS_DECIMAL"
//
// Digit Control Messages
//
#define DIGM_SET          (WM_USER+1)
```

```
#define DIGM_INC          (WM_USER+3)
#define DIGM_DEC          (WM_USER+4)
#define DIGM_SETBASE      (WM_USER+5)
#define DIGM_GETBASE      (WM_USER+6)
#define DIGM_BLANK        (WM_USER+7)
//
// Digit Notification Messages
//
#define DIGN_CARRY        1
#define DIGN_BORROW       2
#define DIGN_LCLICK       3
#define DIGN_RCLICK       4
//
// IDs for Style and About dialog boxes used with the dialog box editors.
//
#define IDD_ABOUT         200

#define IDD_BLANK         101
#define IDD_COLON         102
#define IDD_DECIMAL       103
#define IDD_LED           104
#define IDD_LCD           105
#define IDD_CTLID         106
```

DIGIT.DEF
Complete Listing

```
;-------------------------------------------------------------------
; DIGIT.DEF module definition file
;
; Copyright (c) Douglas Boling 1992
;-------------------------------------------------------------------

LIBRARY              DIGIT

DESCRIPTION          'Digit Control, Copyright 1992, Douglas Boling'
EXETYPE              WINDOWS
CODE                 PRELOAD MOVEABLE
DATA                 PRELOAD MOVEABLE
HEAPSIZE             8192

EXPORTS         LIBMAIN
                WEP             @1
                DIGITINFO       @2
                DIGITSTYLE      @3
                DIGITFLAGS      @4
                DIGITWNDFN      @5
                DIGITDLGFN      @6
```

PC MAGNET

Figure 4: **This is a link definition file that assigns the ordinal numbers for the DigitClass procedures.**

for the DigitClass window and is identical to what would have been written had the control been implemented as a window class inside the program. DigitWndFn handles window messages sent to the window, such as WM_CREATE, WM_PAINT, and WM_COMMAND. It also must manage WM_USER messages that have been equated with control commands. In the case of DigitClass, it deals with DIGM_SET and the other command messages relayed to the control.

DigitWndFn also sends the WM_COMMAND messages containing the notification codes. These messages are sent to the parent of the DigitClass window, as determined by a call to the GetParent Windows API function. Notification messages are sent out in response to the conditions mentioned above. For example, when DigitWndFn receives a WM_LBUTTONUP message, it sends the notification DIGN_LCLICK to the control's parent window.

A series of GDI MoveTo and LineTo calls displays the LED segments in response to the WM_PAINT message. The DigitClass window's dimensions determine the position and width of the lines.

Dialog-box editors use the remaining DIGIT.DLL procedures. Both the SDK's editor (DLGEDIT.EXE) and the Resource Workshop bundled with the Borland C++ 3.0 compiler (WORKSHOP.EXE) can load custom controls that use the custom control interface employed by DigitClass. The routines in the custom control DLL allow these editors to use and display the controls without knowing how they work or even what they look like! To learn how to load and use the DigitClass control in DLGEDIT and WORKSHOP, read the sidebar "Using the DigitClass Control."

The DigitInfo procedure allows a dialog-box editor to query the control for information, such as the copyright, the version number, the control's name, and its default size and style. The procedure itself is simple. It allocates global memory for the information structure, fills in the structure, and then passes the handle back to the editor (which must release the memory block after reading the information).

The dialog-box editor calls DigitStyle to instruct the custom-control DLL to display the control's Style dialog box. Most dialog-box editors call this routine when the user double-clicks on the control. Suppose a user double-clicks on a DigitClass control (to set its styles) just placed on the dialog box being created. In DigitClass, the DigitStyle routine generates a procedure instance for the Style dialog-box procedure, displays the box using Windows' DialogBox function, then frees the instance (with FreeProcInstance) once the function returns.

When DigitStyle is called, a word contains a handle to a block of memory that contains the control's current style structure. The style double-word (DWORD) in this structure contains the current style settings. The dialog box is charged with the job of querying the user aboout any style changes and passing the new control style back to the editor in the style DWORD.

The window procedure for the style dialog box (DigitDlgFn) is a standard dialog-box procedure that responds to notification messages from the dialog-box controls. The dialog-box controls are defined in the Style dialog-box template bound with the DigitClass DLL. When the user presses the OK button on the Style dialog, its procedure updates the control's Style DWORD and destroys the dialog box with a call to EndDialog. Conversely, if the user presses the Cancel button, the Style DWORD is passed back to the dialog-box editor unmodified.

The DigitFlags procedure translates the state of the style bits into the labels used in DIGIT.H. The labels are then written into the resulting .DLG file for the dialog box. When called, the routine is passed three parameters: the Style DWORD, a pointer to an ASCIIZ string, and the maximum length of that string. DigitFlags must examine the Style DWORD and append the appropriate style tags to the ASCIIZ string. For example, if the Style DWORD has the LED display bit set, DigitFlags would append the DIGS_LED string to the ASCIIZ string. The dialog-box editor would then use the resulting string in the .DLG file's CONTROL statement that specifies the DigitClass control.

The procedures we've discussed provide everything a dialog-box editor needs to create a CONTROL statement in the dialog template. The editor can determine the name of the control class by calling the DigitInfo procedure. The dialog editor can display a custom style dialog box for the control by calling DigitStyle. When the user moves the control, its position and size are determined by the dialog-box editor. Finally, the editor calls the DigitFlags routine to convert the style bits into the labels used in the .DLG file. This combination of routines allows the dialog-box editor to produce the .RES and .DLG files.

DigitClass furnishes a simple example, but it demonstrates all the necessary routines for a custom control. It also demonstrates all the necessary routines for a custom control. And, it demonstrates that the ideas behind the standard controls can be applied to controls customized for specific applications. □

DOUGLAS BOLING IS A CONTRIBUTING EDITOR TO *PC MAGAZINE.*

DIGIT.DEF
Complete Listing

```
;----------------------------------------------------------------------
; DIGIT.DEF module definition file
;
; Copyright (c) Douglas Boling 1992
;----------------------------------------------------------------------

LIBRARY                 DIGIT

DESCRIPTION             'Digit Control, Copyright 1992, Douglas Boling'
EXETYPE                 WINDOWS
CODE                    PRELOAD MOVEABLE
DATA                    PRELOAD MOVEABLE
HEAPSIZE                8192

EXPORTS            LIBMAIN
                   WEP             @1
                   DIGITINFO       @2
                   DIGITSTYLE      @3
                   DIGITFLAGS      @4
                   DIGITWNDFN      @5
                   DIGITDLGFN      @6
```

Figure 4: **This is a link definition file that assigns the ordinal numbers for the DigitClass procedures.**

for the DigitClass window and is identical to what would have been written had the control been implemented as a window class inside the program. DigitWndFn handles window messages sent to the window, such as WM_CREATE, WM_PAINT, and WM_COMMAND. It also must manage WM_USER messages that have been equated with control commands. In the case of DigitClass, it deals with DIGM_SET and the other command messages relayed to the control.

DigitWndFn also sends the WM_COMMAND messages containing the notification codes. These messages are sent to the parent of the DigitClass window, as determined by a call to the GetParent Windows API function. Notification messages are sent out in response to the conditions mentioned above. For example, when DigitWndFn receives a WM_LBUTTONUP message, it sends the notification DIGN_LCLICK to the control's parent window.

A series of GDI MoveTo and LineTo calls displays the LED segments in response to the WM_PAINT message. The DigitClass window's dimensions determine the position and width of the lines.

Dialog-box editors use the remaining DIGIT.DLL procedures. Both the SDK's editor (DLGEDIT.EXE) and the Resource Workshop bundled with the Borland C++ 3.0 compiler (WORKSHOP.EXE) can load custom controls that use the custom control interface employed by DigitClass. The routines in the custom control DLL allow these editors to use and display the controls without knowing how they work or even what they look like! To learn how to load and use the DigitClass control in DLGEDIT and WORKSHOP, read the sidebar "Using the DigitClass Control."

The DigitInfo procedure allows a dialog-box editor to query the control for information, such as the copyright, the version number, the control's name, and its default size and style. The procedure itself is simple. It allocates global memory for the information structure, fills in the structure, and then passes the handle back to the editor (which must release the memory block after reading the information).

The dialog-box editor calls DigitStyle to instruct the custom-control DLL to display the control's Style dialog box. Most dialog-box editors call this routine when the user double-clicks on the control. Suppose a user double-clicks on a DigitClass control (to set its styles) just placed on the dialog box being created. In DigitClass, the DigitStyle routine generates a procedure instance for the Style dialog-box procedure, displays the box using Windows' DialogBox function, then frees the instance (with FreeProcInstance) once the function returns.

When DigitStyle is called, a word contains a handle to a block of memory that contains the control's current style structure. The style double-word (DWORD) in this structure contains the current style settings. The dialog box is charged with the job of querying the user aboout any style changes and passing the new control style back to the editor in the style DWORD.

The window procedure for the style dialog box (DigitDlgFn) is a standard dialog-box procedure that responds to notification messages from the dialog-box controls. The dialog-box controls are defined in the Style dialog-box template bound with the DigitClass DLL. When the user presses the OK button on the Style dialog, its procedure updates the control's Style DWORD and destroys the dialog box with a call to EndDialog. Conversely, if the user presses the Cancel button, the Style DWORD is passed back to the dialog-box editor unmodified.

The DigitFlags procedure translates the state of the style bits into the labels used in DIGIT.H. The labels are then written into the resulting .DLG file for the dialog box. When called, the routine is passed three parameters: the Style DWORD, a pointer to an ASCIIZ string, and the maximum length of that string. DigitFlags must examine the Style DWORD and append the appropriate style tags to the ASCIIZ string. For example, if the Style DWORD has the LED display bit set, DigitFlags would append the DIGS_LED string to the ASCIIZ string. The dialog-box editor would then use the resulting string in the .DLG file's CONTROL statement that specifies the DigitClass control.

The procedures we've discussed provide everything a dialog-box editor needs to create a CONTROL statement in the dialog template. The editor can determine the name of the control class by calling the DigitInfo procedure. The dialog editor can display a custom style dialog box for the control by calling DigitStyle. When the user moves the control, its position and size are determined by the dialog-box editor. Finally, the editor calls the DigitFlags routine to convert the style bits into the labels used in the .DLG file. This combination of routines allows the dialog-box editor to produce the .RES and .DLG files.

DigitClass furnishes a simple example, but it demonstrates all the necessary routines for a custom control. It also demonstrates all the necessary routines for a custom control. And, it demonstrates that the ideas behind the standard controls can be applied to controls customized for specific applications. □

DOUGLAS BOLING IS A CONTRIBUTING EDITOR TO *PC MAGAZINE.*